CGT 353: Principles of Interactive and Dynamic Media

XML

Overview

XML is:

- Human readable and editable
- Standard format that web languages understand
- Easy to transfer data from one language to another
- Easily stored and modified

XML basics and terms

Terms

1. **XML document** - A file that contains XML. This term may also refer to XML data that is being loaded or sent. An XML document is not to be confused with the *XMLDocument* class.
2. **XML packet** - An XML packet can be any snippet of XML—from an entire XML document to just a single node—as long as it is represents a complete, well-formed piece of information in XML.
3. **XML node** - The basic building block for XML. Nodes can be elements, text nodes, attributes, and so on. We refer to elements and text nodes collectively as "nodes" when talking in general terms.
4. **XML element** - The term "element" is often used interchangeably with the term "tag." More accurately, however, an element contains tags. Elements must have an opening and closing tag (<element></element>), or the opening and closing tags can be combined into one when the element has no nested elements (<element />).
5. **Root node** - An element that is at the top of the XML hierarchy of elements.
6. **Text node** - A node containing text. Text nodes are generally nested within elements.
7. **Attribute** - Is part of an element. Attributes are placed within the tags of elements in name/value pair format, such as <element name="value">.
8. **XML declaration** - The declaration typically looks like this: <?xml version="1.0" ?>. This is a special tag that the XML parser recognizes as containing information about the XML document, and it is not parsed as an element.
9. **XML tree** - Also sometimes called the "data tree," an XML tree is the hierarchy of nodes in XML data.

Basics

- XML is simple text, with 'tags' enclosing the data
- Each tag and the data within it is called a node
- Well-formedness: must close the tags in the order they were opened
- Child: a node that is contained within another node
- Parent: the node that acts as the container for the current node

```
<BOOK ISBN="0141182806">
        <TITLE>Ulysses</TITLE>
        <AUTHOR>Joyce, James</AUTHOR>
        <PUBLISHER>Penguin Books Ltd</PUBLISHER>
</BOOK>
```

- What is the first node in book?
    - Whitespace

- In E4X, whitespace counts:
    - carriage return (\u000D)
    - space (\u0020)
    - line feed (\u000A)
    - tab (\u0009)
- So technically <BOOK> has seven nodes, not four.
- By default whitespace nodes are ignored by the E4X parser, so we don't really have to worry about it.

- CREATE A XML FILE HERE -

XML in actionscript

Actionscript 2.0

- XML Class
    - W3C Document Object Model (DOM)
    - Ability to loop though children, acess parents
    - These operators are still in 3.0
    - Really only good for certain types or uses of XML
        - what kinds?

Actionscript 3.0

- ECMAscript for XML (E4X)
- The official standard for working with XML for all ECMAscripting languages (Actionscript, Javascript)

Data Representation in E4X

- Two datatypes: XML, and XMLList
- Five types of content (or nodes)
    - Elements
    - Attributes
    - Text nodes
    - Comments
    - Processing instructions

- Any child nodes of an element are wrapped in an XMLList

```
<BOOK ISBN="0141182806">
        <TITLE>Ulysses</TITLE>
        <AUTHOR>Joyce, James</AUTHOR>
        <PUBLISHER>Penguin Books Ltd</PUBLISHER>
</BOOK>
```

- <BOOK> contains **two** *XMLList* instances—one for <BOOK>'s attributes and the other for its child elements.
- The <BOOK> element has only one attribute, so the *XMLList* for <BOOK>'s attributes contains one *XML* instance only

Creating XML Data with E4X:

Three Options:

1. Use the *XML* constructor to create a new *XML* instance, then create the remainder of the fragment programmatically
2. Use the *XML* constructor to create a new *XML* instance, then import the fragment from an externally loaded file
3. Write our *XML* data in literal form, just like a string or a number, anywhere literals are allowed by ActionScript.

Option 3:

```
var novel:XML = <BOOK ISBN="0141182806">
                    <TITLE>Ulysses</TITLE>
                    <AUTHOR>Joyce, James</AUTHOR>
                    <PUBLISHER>Penguin Books Ltd</PUBLISHER>
        </BOOK>;
```

- Notice that the line breaks and quotation marks are normal. ActionScript knows they are part of the XML data.
- ActionScript also allows dynamic expressions to be used so that element names, attribute names, attribute values, and element content can be generated programmatically.
- To specify a dynamic expression within an XML literal, surround it in curly braces ({ }).

Example 1.

```
var elementName:String = "BOOK";
var novel:XML = <{elementName}/>;
```

Example 2.

```
var rootElementName:String = "BOOK";
var rootAttributeName:String = "ISBN";
var childElementNames:Array = ["TITLE", "AUTHOR", "PUBLISHER"];
var bookISBN:String = "0141182806";
var bookTitle:String = "Ulysses";
var bookAuthor:String = "Joyce, James";
var bookPublisher:String = "Penguin Books Ltd";
var novel:XML = <{rootElementName} {rootAttributeName}={bookISBN}>
                        <{childElementNames[0]}>{bookTitle}</{childElementNames[0]}>
<{childElementNames[1]}>{bookAuthor}</{childElementNames[1]}>
```

```
<{childElementNames[2]}>{bookPublisher}</{childElementNames[2]}>
        </{rootElementName}>;
```

Example 3.

```
        // Create the XML structure with a string so use the value of both
        // username and score inside of the XML packet.
        var str:String = "<gamescore><username>" + username + "</username>"
+ "<score>" + score + "</score></gamescore>";

        // Pass the string to the constructor to create an XML object
        var example:XML = new XML( str );
```

Acessing the data

Two General Sets of Tools:

1. The *XML* and *XMLList* content-access methods (*attribute( )*, *attributes( )*, *child( )*, *children( )*, *comments( )*, *descendants( )*, *elements( )*, *parent( )*, *processingInstructions( )*, and *text( )*)

2. Variable-style access with the dot (.), descendant (..), and attribute (@) operators

☐ Variable-style access is offered as a convenience and always equates to one of the methods of either the *XML* or *XMLList* classes.

☐ The two approaches do not overlap completely.

☐ The following types of content must be accessed using the appropriate method of the *XML* or *XMLList* class:

1. An*XML* instance's parent (accessed via *parent( )*)
2. Comments (accessed via *comments( )*)
3. Processing instructions (accessed via *processingInstructions( )*)
4. Elements or attributes whose names include characters considered illegal in an
5. ActionScript identifier (accessed via *attribute( )*, *child( )*, *descendants( )*, or*elements( )*)

Accessing Child Nodes:

- Use the children() function to access child nodes

  ```
  novel.children( ) // Returns an XMLList representing <BOOK>'s child nodes

  novel.* // Also returns an XMLList, representing <BOOK>'s child node
  ```

  To access a specific child in an *XMLList* we use the familiar array-element access operator, [].

  ```
  novel.children( )[1] // A reference to <BOOK>'s second child node
  ```

  or:

```
novel.*[1] // Also a reference to <BOOK>'s second child node
```

- There is no firstChild or lastChild variable in E4X
- First child in a list of child nodes can be accessed as follows:

theNode.children( )[0]

And the last child in a list of child nodes can be accessed as follows:

theNode.children( )[*theNode*.children().length( )-1]

- Accessing child nodes according to its position in a list can be cumbersome, and has been deemphasized by E4X.
- In E4X, child nodes are typically accessed by their element names rather than their position.
- To retrieve an *XMLList* of all children of <BOOK> named "AUTHOR", we use:

```
novel.child("AUTHOR") // Returns all child elements of <BOOK> named "AUTHOR"

or, using variable access syntax….

novel.AUTHOR // Also returns all child elements of <BOOK> named "AUTHOR"
```

Ex 2.

```
var novel:XML = <BOOK>

<AUTHOR>Jacobs, Tom</AUTHOR>

<AUTHOR>Schumacher, Jonathan</AUTHOR>

</BOOK>;

novel.AUTHOR[0]; // Access <AUTHOR>Jacobs, Tom</AUTHOR>

novel.AUTHOR[1]; // Access <AUTHOR>Schumacher, Jonathan</AUTHOR>
```

Accessing Text Nodes:

```
var novel:XML = <BOOK ISBN="0141182806">

<TITLE>Ulysses</TITLE>

<AUTHOR>Joyce, James</AUTHOR>

<PUBLISHER>Penguin Books Ltd</PUBLISHER>

</BOOK>;

novel.TITLE.children( )[0] // A reference to the text node Ulysses
```

Or, alternatively, we can use the properties wildcard to do the same thing:

```
novel.TITLE.*[0] // Also a reference to the text node Ulysses

novel.TITLE.*[0].parent( ) // Reference to the <TITLE> element
novel.TITLE.*[0].nodeKind( ) // Returns the string "text"

novel.TITLE.*[0].toString( ) // Returns the string "Ulysses"
```

Accessing Parent Nodes:

```
var pub:XML = novel.PUBLISHER[0];
```

- To access <PUBLISHER>'s parent (which is <BOOK>), we use:
  `pub.parent( )`

- The *parent( )* method can also be used successively to access any ancestor node:

```
// Create a 3-tier XML hierarchy.

var doc:XML = <grandparent><parent><child></child></parent></grandparent>;

// Assign a reference to <child>

var kid:XML = doc.parent.child[0];

// Use parent( ) successively to access <grandparent> from <child>

        var grandparent:XML = kid.parent().parent( );
```

Accessing Sibling Nodes:

- A sibling node is a node that resides directly beside another node on a given level of an XML hierarchy
- E4X reduces the emphasis on accessing siblings due to its increased focus on accessing elements by name.

```
var novel:XML = <BOOK ISBN="0141182806">

<TITLE>Ulysses</TITLE> <!--Previous sibling-->

<AUTHOR>Joyce, James</AUTHOR>

<PUBLISHER>Penguin Books Ltd</PUBLISHER> <!--Next sibling-->

</BOOK>;
```

- In E4X there is no built-in support for moving between sibling nodes in an XML hierarchy.
- The DOM-based nextSibling, previousSibling variables are not part of the E4X API
- What should you use?

Next sibling:

someNode.parent( ).*[*someNode*.childIndex( )+1];

Previous sibling:

```
someNode.parent( ).*[someNode.childIndex( )-1];
```

Accessing and Adding Attributes:

- To access an *XMLList* representing all of an element's attributes, we use the *XML* class's instance method *attributes( )*
- Alternatively, we can access an *XMLList* representing an element's attributes using the E4X *attributes wildcard* (@*)

```
someElement.@* // Returns an XMLList representing all of someElement's attributes
```

- Use the @ operator in E4X syntax to assign attributes to element nodes.

```
// Create an XML instance to work with

var example:XML = <example><someElement/></example>;

// Add some attributes to the someElement element node

example.someElement.@number = 12.1;

example.someElement.@string = "example";

example.someElement.@boolean = true;

example.someElement.@array = ["a", null, 7, undefined, "c"];



/* Displays:

<example>

  <someElement number="12.1" string="example" boolean="true"

  array="a,,7,,c"/>

</example>

*/

trace( example );
```

* ACESS THE DATA ALREADY MADE *